Programming Models for Blue Gene/L : Charm++, AMPI and Applications

Laxmikant (Sanjay) Kale
Parallel Programming Laboratory
Dept. of Computer Science
University of Illinois at Urbana Champaign
http://charm.cs.uiuc.edu

Acknowlwdgements

- Graduate students including:
  - Gengbin Zheng
  - Orion Lawlor
  - Milind Bhandarkar
  - Arun Singla
  - Josh Unger
  - Terry Wilmarth
  - Sameer Kumar
- Recent Funding:
  - NSF (NGS: Frederica Darema)
  - DOE (ASCI : Rocket Center)
  - NIH (Molecular Dynamics)

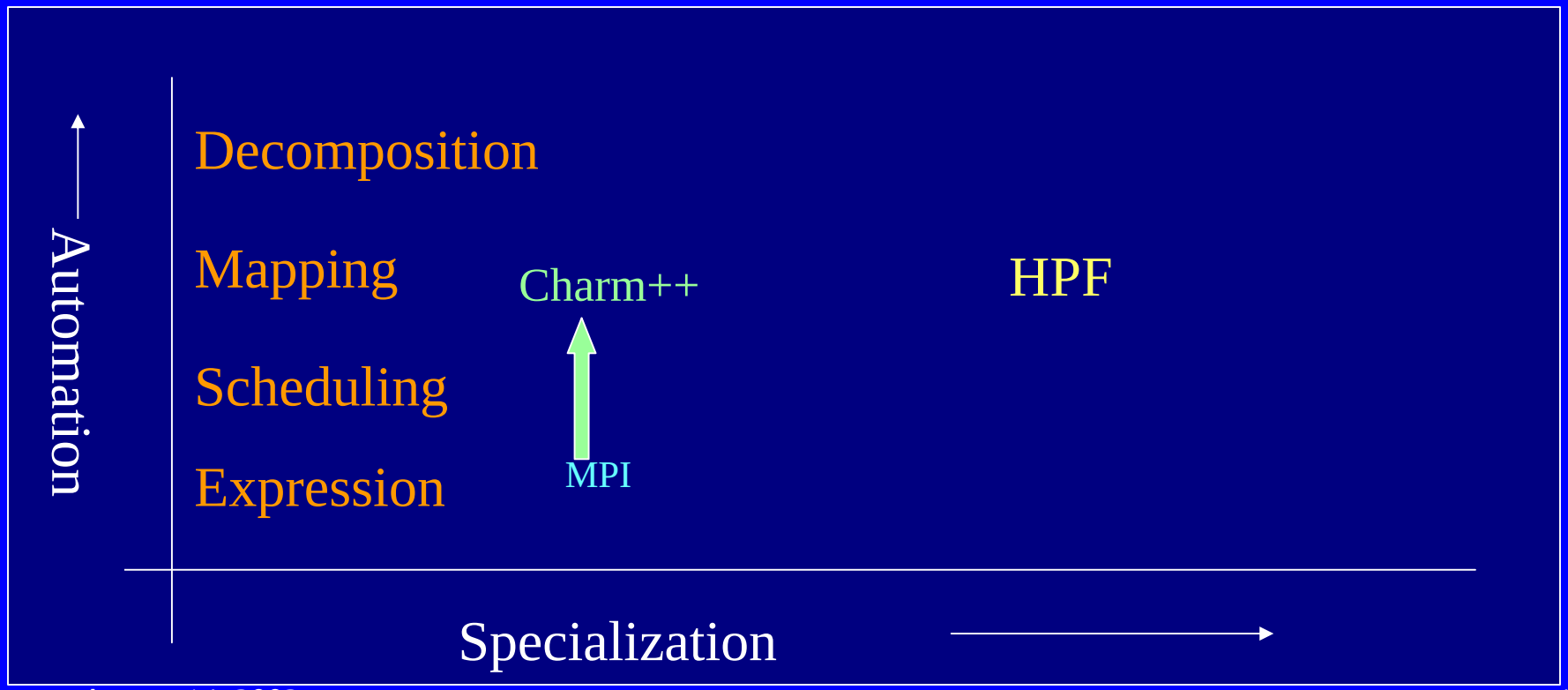
Outline

- The virtualization model
  - Charm++ and AMPI
  - Virtualization: a silver bullet
- BG/L Program Development environment
  - Emulation setup
  - Simulation and Performance Prediction
- Applications using BG/L
  - Scaling Issues
  - Example: Molecular Dynamics
- Ongoing research

Technical Approach

- Seek optimal division of labor between “system” and programmer:

Decomposition done by programmer, everything else automated

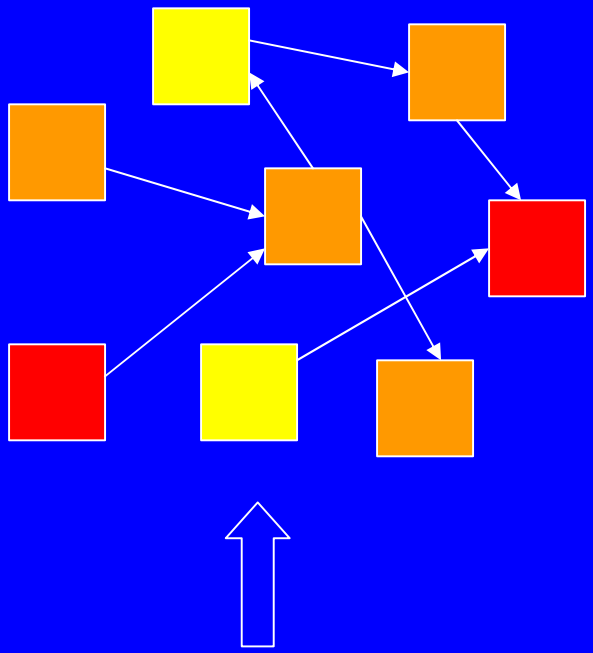


Object-based Decomposition

- Basic Idea:
  - Divide the computation into a large number of pieces
    - Independent of number of processors
    - Typically larger than number of processors
  - Let the *system* map objects to processors
- Old idea? G. Fox Book ('86?), DRMS (IBM), ..
- Our approach is *"virtualization++"*
  - Language and runtime support for virtualization
  - Exploitation of virtualization to the hilt

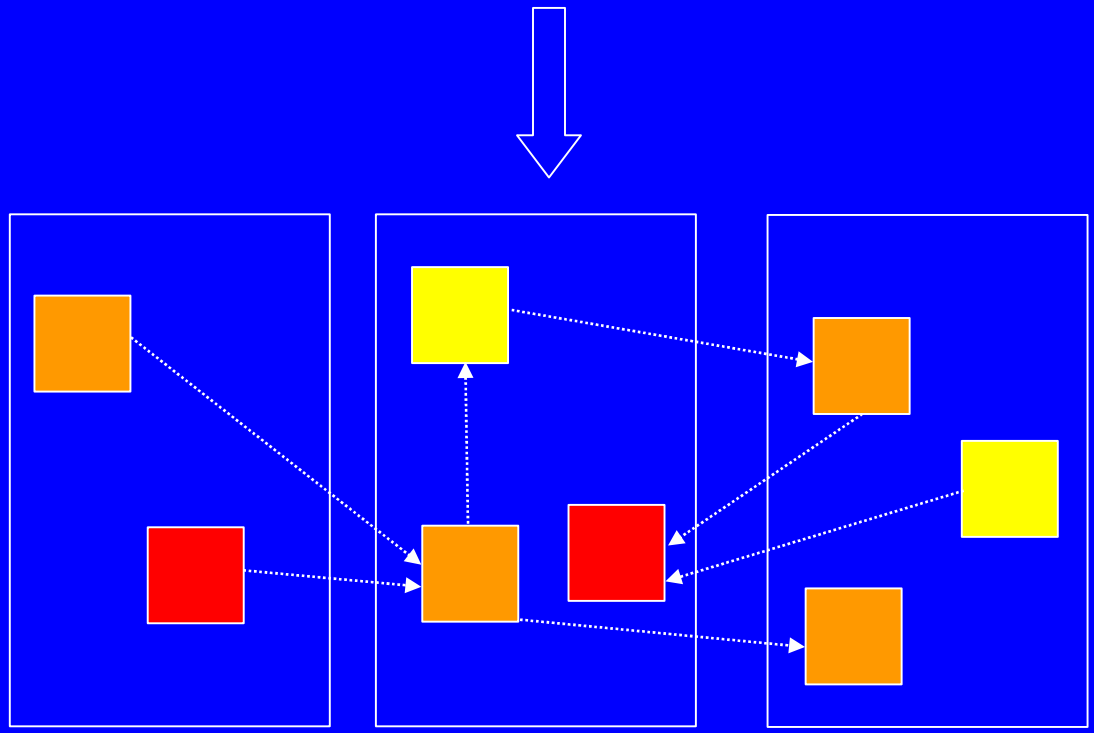
Virtualization: Object-based Parallelization

User is only concerned with interaction between objects (VPs)



User View

System implementation

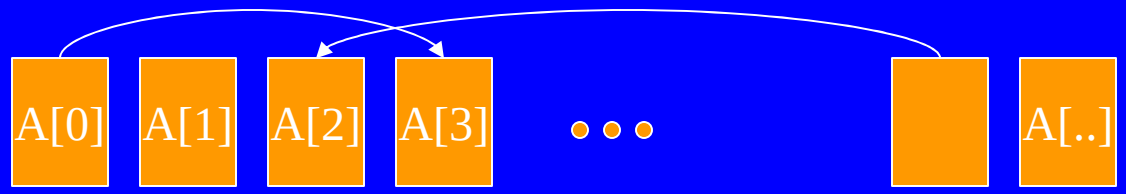


Realizations: Charm++

- Charm++
  - Parallel C++ with Data Driven Objects (Chares)
  - Asynchronous method invocation
    - Prioritized scheduling
  - Object Arrays
  - Object Groups:
  - Information sharing abstractions: readonly, tables,..
  - Mature, robust, portable (http://charm.cs.uiuc.edu)

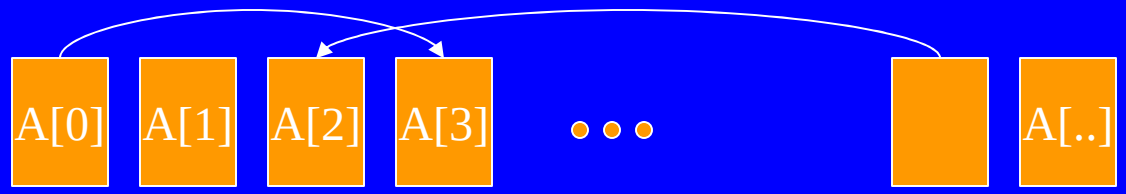
Object Arrays

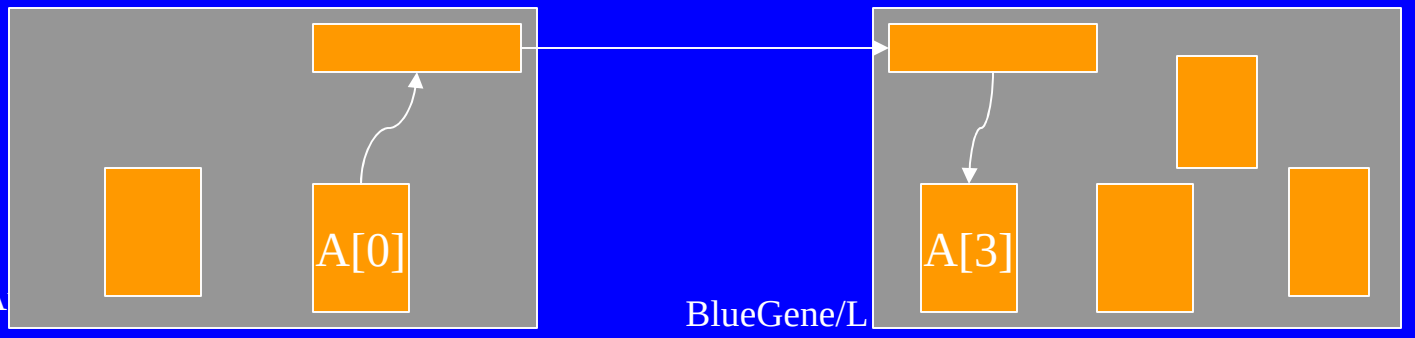
- A collection of data-driven objects
  - With a single global name for the collection
  - Each member addressed by an index
    - [sparse] 1D, 2D, 3D, tree, string, ...
  - Mapping of element objects to procS handled by the system



Object Arrays

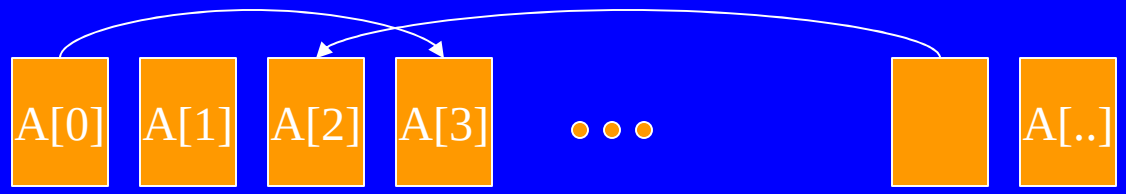
- A collection of data-driven objects
  - With a single global name for the collection
  - Each member addressed by an index
    - [sparse] 1D, 2D, 3D, tree, string, ...
  - Mapping of element objects to procS handled by the system

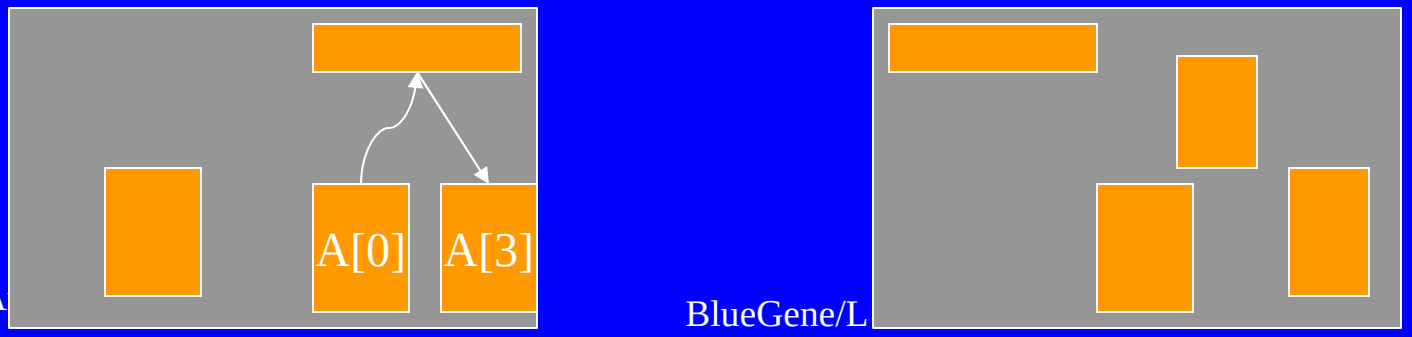




Object Arrays

- A collection of data-driven objects
  - With a single global name for the collection
  - Each member addressed by an index
    - [sparse] 1D, 2D, 3D, tree, string, ...
  - Mapping of element objects to procS handled by the system





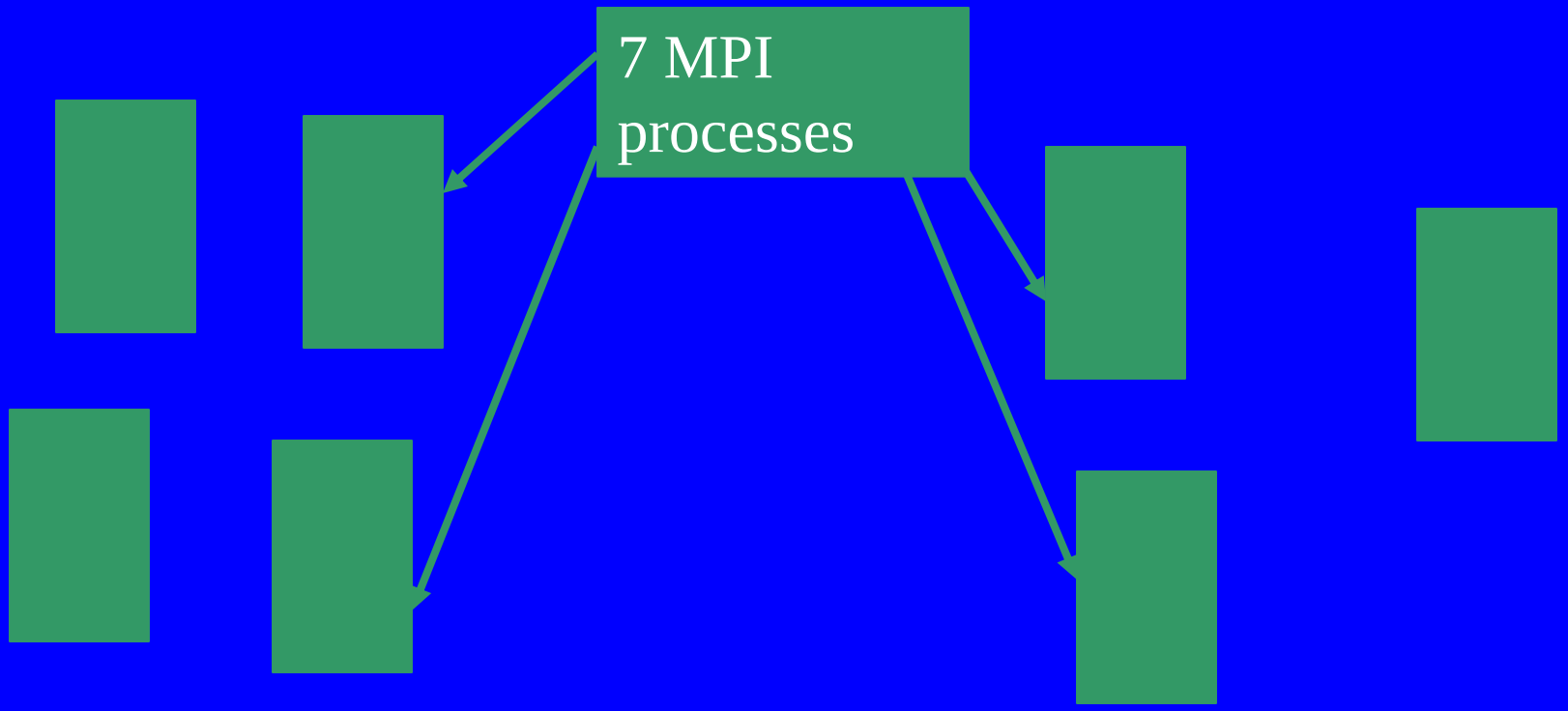
Comparison with MPI

- Advantage: Charm++
  - Modules/Abstractions are centered on application data structures,
    - Not processors
  - Several other…
- Advantage: MPI
  - Highly popular, widely available, industry standard
  - *"Anthropomorphic"* view of processor
    - Many developers find this intuitive
- But mostly:
  - There is no hope of weaning people away from MPI
  - There is no need to choose between them!

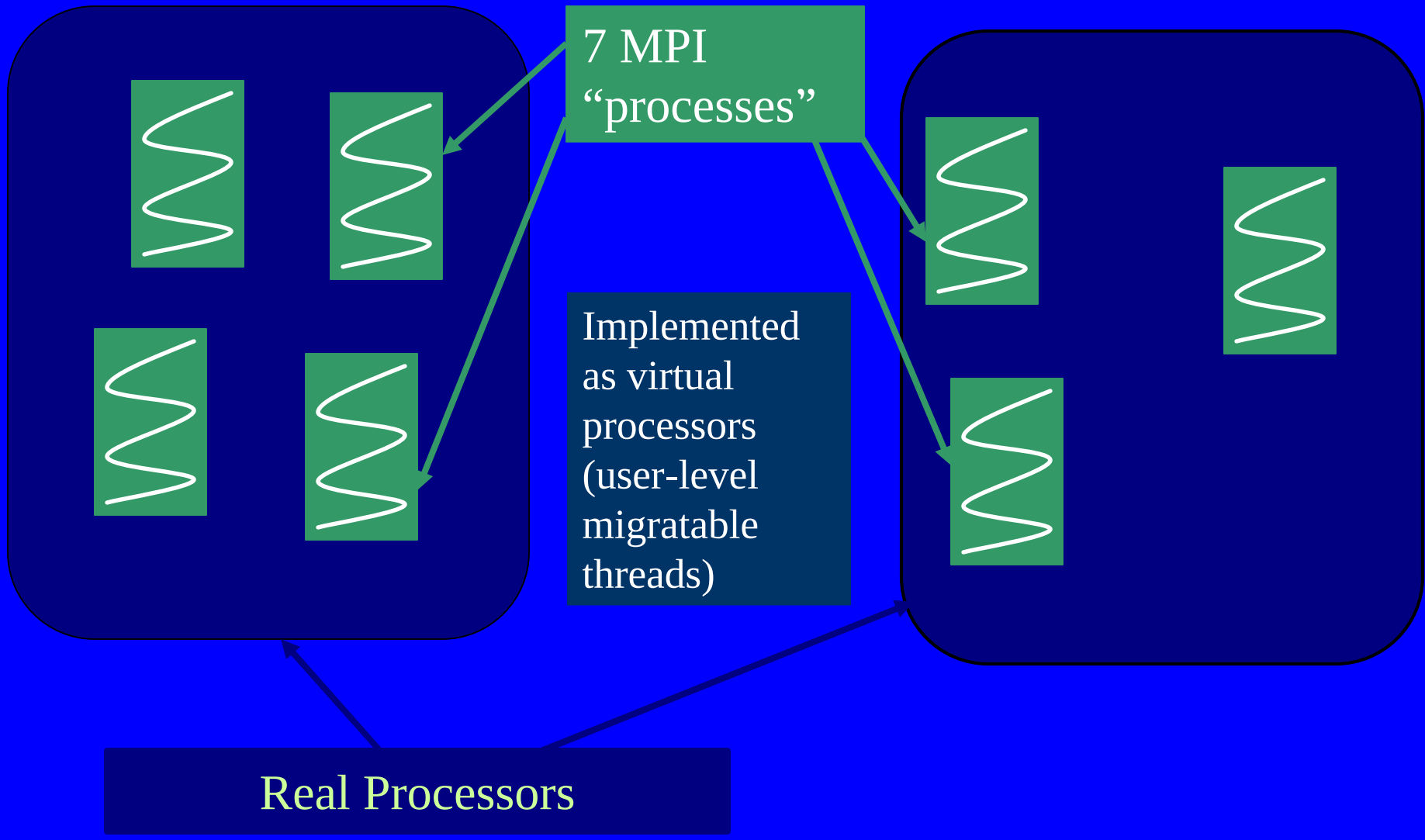
Adaptive MPI

- A migration path for legacy MPI codes
- AMPI = MPI + Virtualization
- Uses Charm++ object arrays and *migratable* threads
- Minimal modifications to convert existing MPI programs
  - Automated via *AMPizer*
    - Based on Polaris Compiler Framework
- Bindings for
  - C, C++, and Fortran90

AMPI:



AMPI:


7 MPI "processes"
Implemented as virtual processors (user-level migratable threads)
Real Processors

II: Benefits of Virtualization

- Better Software Engineering
- Message Driven Execution
- Flexible and dynamic mapping to processors
- Principle of Persistence:
  - Enables Runtime Optimizations
  - Automatic Dynamic Load Balancing
  - Communication Optimizations
  - Other Runtime Optimizations

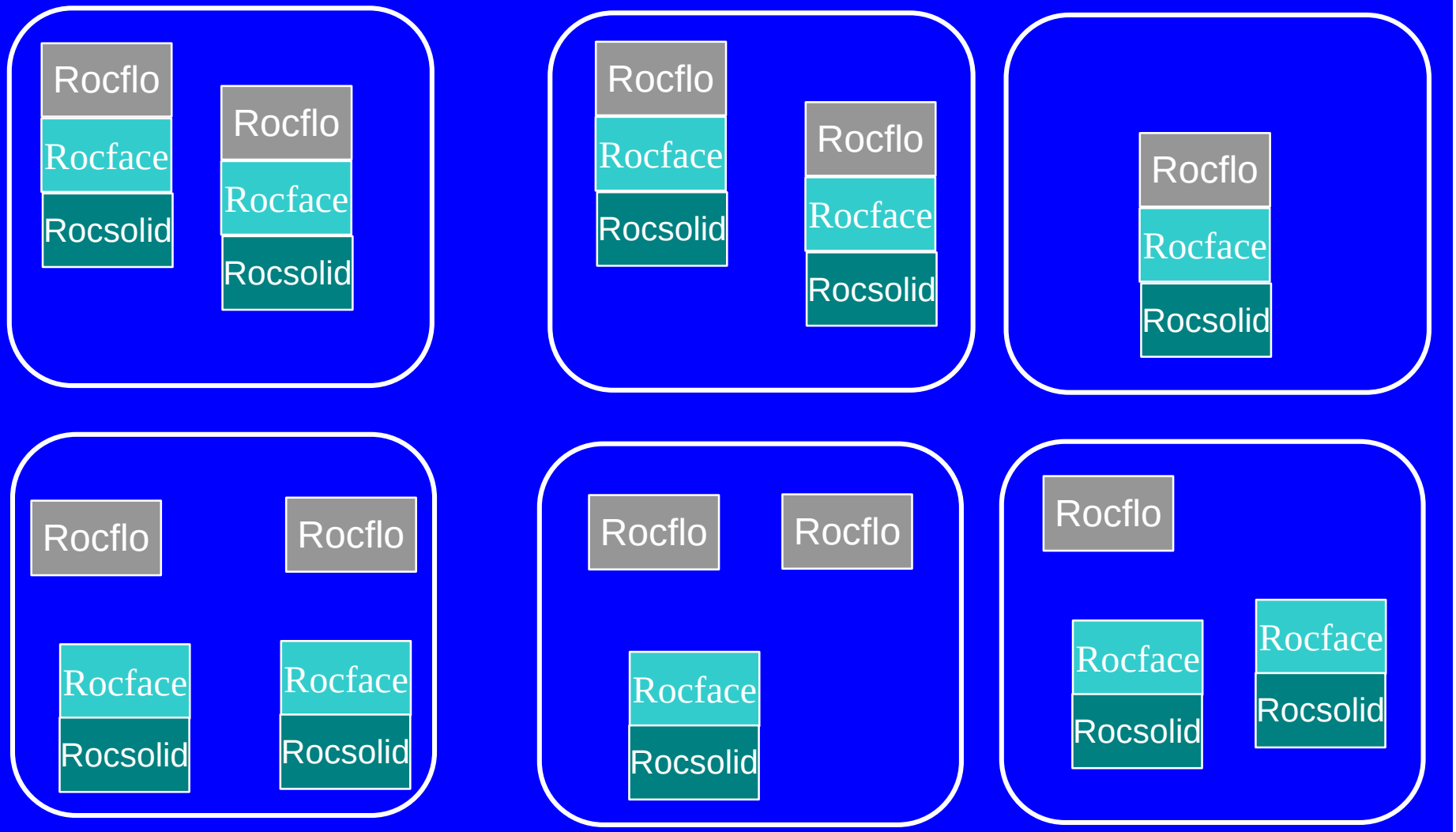
Modularization

- Logical Units decoupled from “Number of processors”
  - E.G. Oct tree nodes for particle data
  - No artificial restriction on the number of processors
    - Cube of power of 2
- Modularity:
  - Software engineering: cohesion and coupling
  - MPI’s “are on the same processor” is a bad coupling principle
  - Objects liberate you from that:
    - E.G. Solid and fluid moldules in a rocket simulation

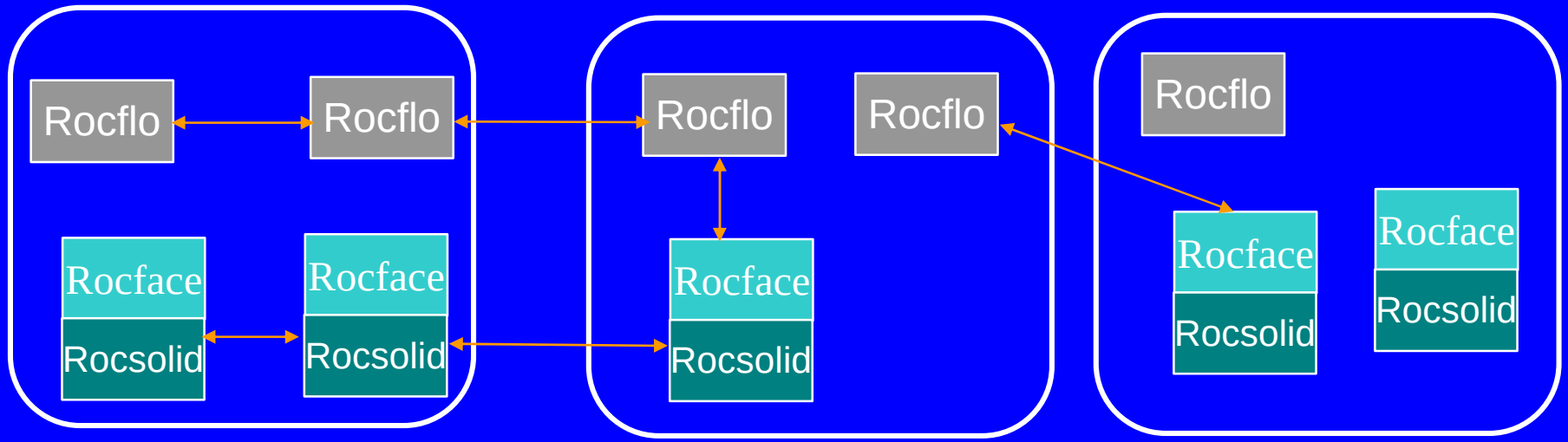
Rocket Simulation

- Large Collaboration headed Mike Heath
  - DOE supported ASCI center
- Challenge:
  - Multi-component code, with modules from independent researchers
  - MPI was common base
- AMPI: new wine in old bottle
  - Easier to convert
  - Can still run original codes on MPI, unchanged

Rocket simulation via virtual processors

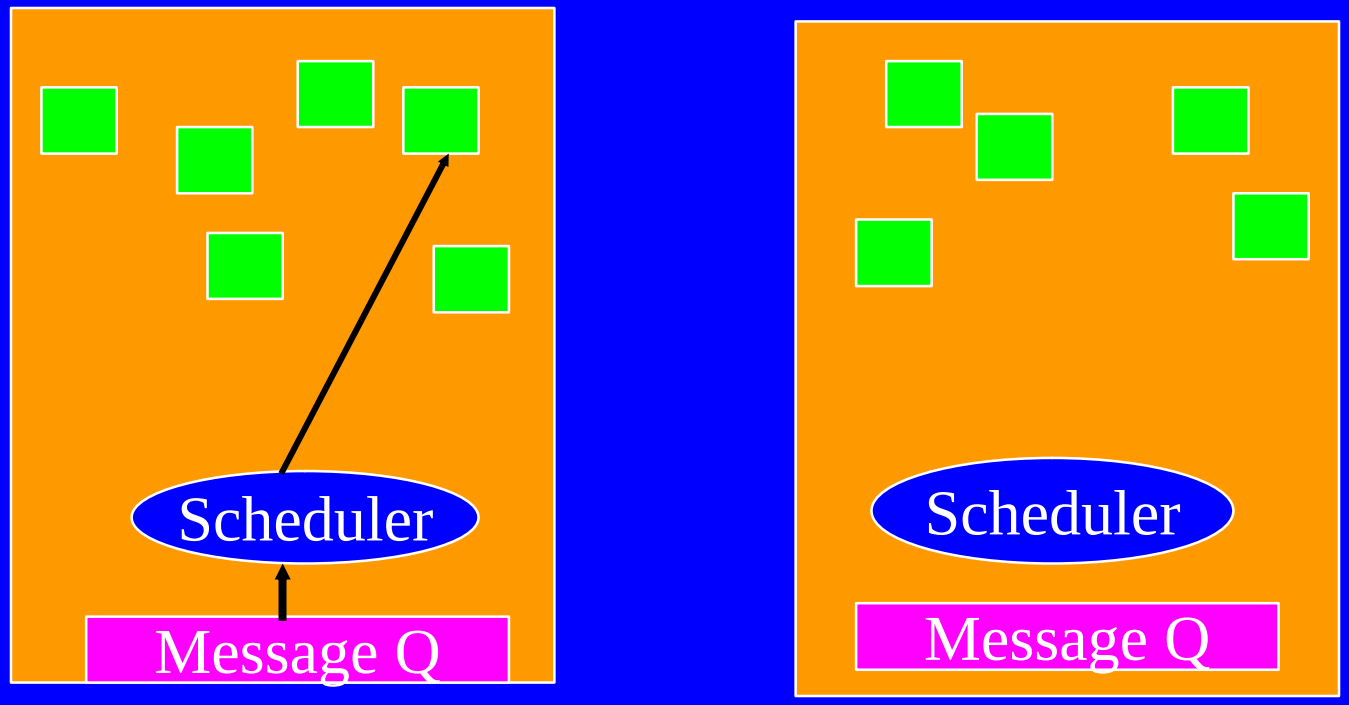


AMPI and Roc*: Communication



Message Driven Execution

Virtualization leads to *Message Driven Execution*

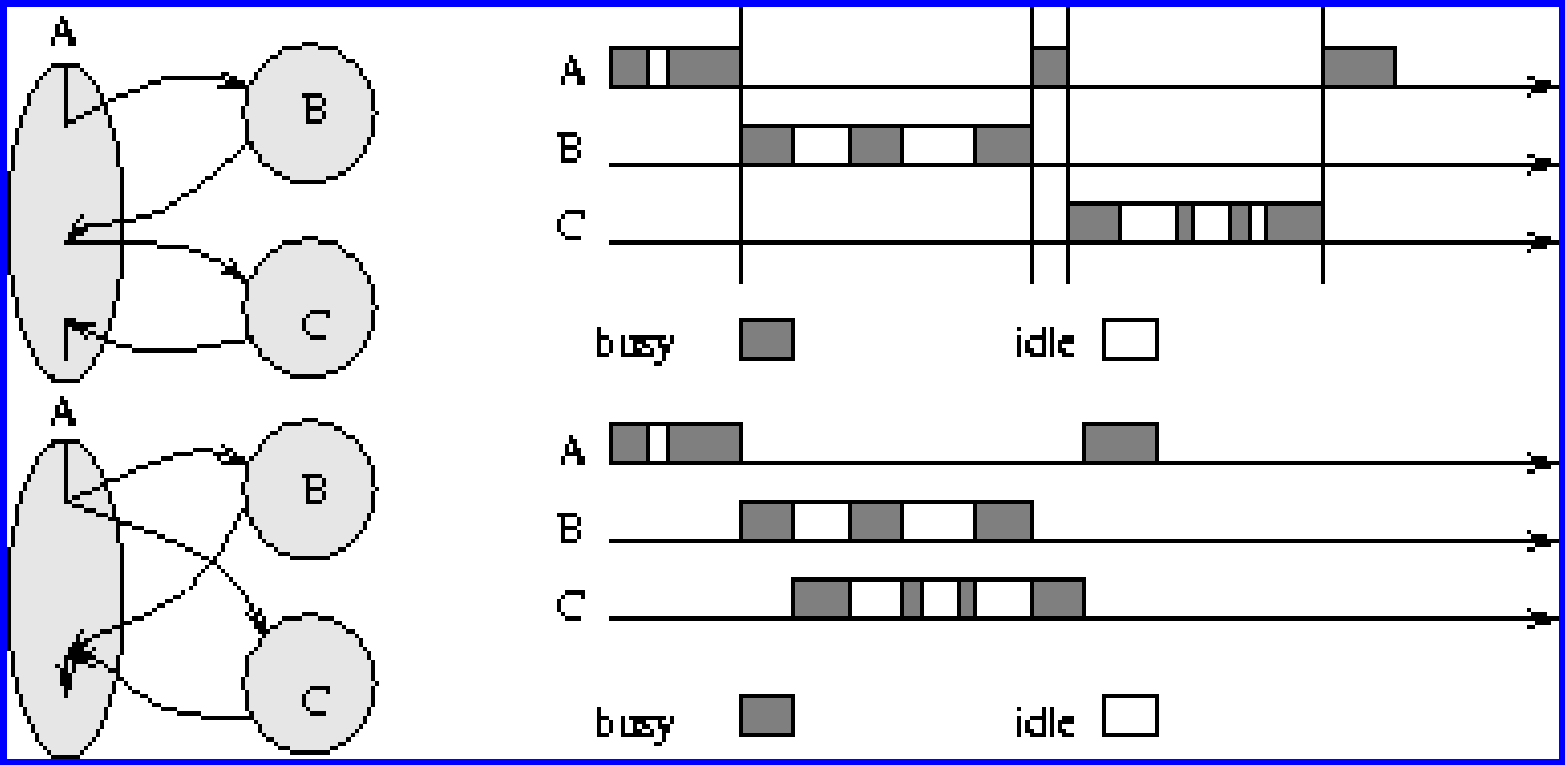


Which leads to Automatic Adaptive overlap of computation and communication

Adaptive Overlap via Data-driven Objects

- Problem:
  - Processors wait for too long at “receive” statements
- Routine communication optimizations in MPI
  - Move sends up and receives down
  - Sometimes. Use irecvs, but be careful
- With Data-driven objects
  - Adaptive overlap of computation and communication
  - No object or threads holds up the processor
  - No need to guess which is likely to arrive first

Adaptive overlap and modules



SPMD and Message-Driven Modules

(From A. Gursoy, *Simplified expression of message-driven programs and quantification of their impact on performance*, Ph.D Thesis, Apr 1994.)

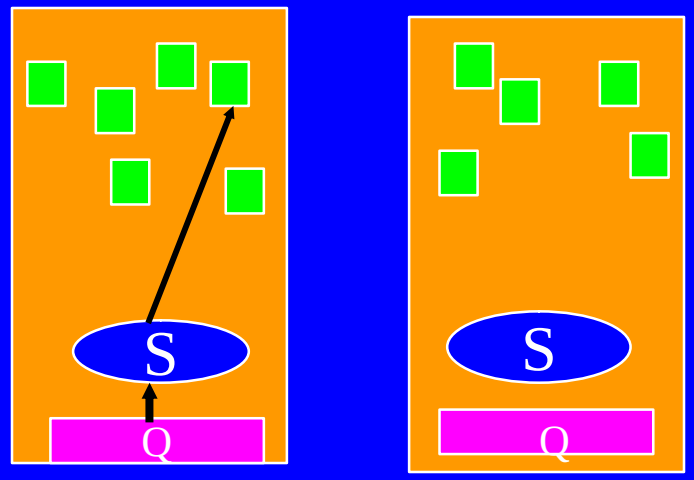
Handling OS Jitter via MDE

- MDE encourages asynchrony
  - Asynchronous reductions, for example
  - Only *data dependence* should force synchronization
- One benefit:
  - Consider an algorithm with N steps
    - Each step has different load balance:*Tij*
    - Loose dependence between steps
      - (on neighbors, for example)
  - Sum-of-max (MPI) vs max-of-sum (MDE)
- OS Jitter:
  - Causes random processors to add delays in each step
  - Handled Automatically by MDE

Virtualization/MDE leads to predictability

- Ability to predict:
  - Which data is going to be needed and
  - Which code will execute
  - Based on the ready queue of object method invocations
- So, we can:
  - Prefetch data accurately
  - Prefetch code if needed
  - Out-of-core execution
  - Caches vs controllable SRAM

Salivating at the shared SRAM in BG/L



Flexible Dynamic Mapping to Processors

- The system can migrate objects between processors
  - Vacate processor used by a parallel program
  - Dealing with extraneous loads on shared workstations
  - Shrink and Expand the set of processors used by an app
    - Shrink from 1000 to 900 procs. Later expand to 1200.
    - Adaptive job scheduling for better System utilization
  - Adapt to speed difference between processors
    - E.g. Cluster with 500 MHz and 1 Ghz processors
- Automatic checkpointing
  - Checkpointing = migrate to disk!
  - Restart on a different number of processors

Principle of Persistence

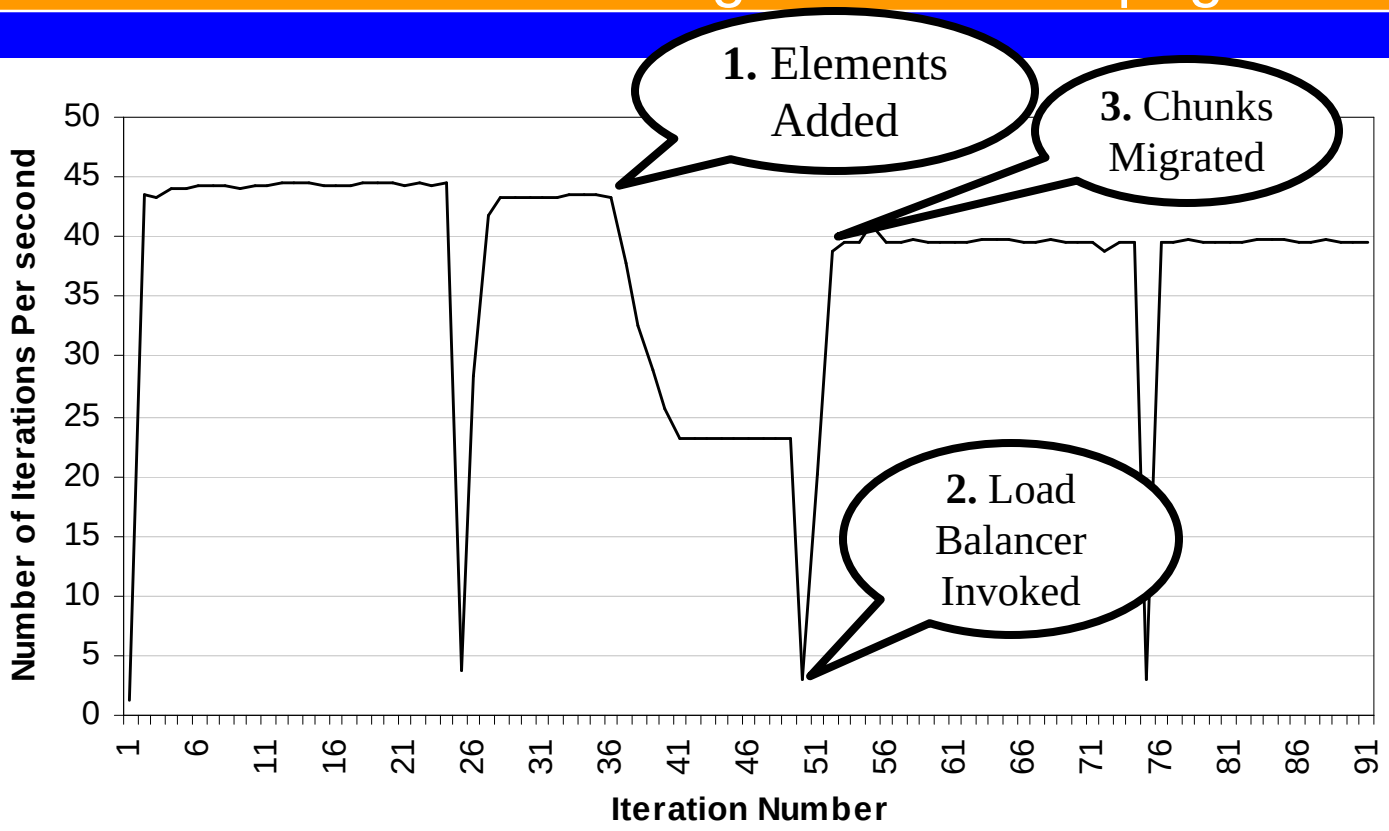
- Once the application is expressed in terms of interacting objects:
  - Object communication patterns and computational loads tend to persist over time
  - In spite of dynamic behavior
    - Abrupt and large,but infrequent changes (eg:AMR)
    - Slow and small changes (eg: particle migration)
- Parallel analog of principle of locality
  - Heuristics, that holds for most CSE applications
  - Learning / adaptive algorithms
  - Adaptive Communication libraries
  - Measurement based load balancing

Measurement Based Load Balancing

- Based on Principle of persistence
- Runtime instrumentation
  - Measures communication volume and computation time
- Measurement based load balancers
  - Use the instrumented data-base periodically to make new decisions
  - Many alternative strategies can use the database
    - Centralized vs distributed
    - Greedy improvements vs complete reassignments
    - Taking communication into account
    - Taking dependences into account (More complex)

Load balancer in action

Automatic Load Balancing in Crack Propagation



Optimizing for Communication Patterns

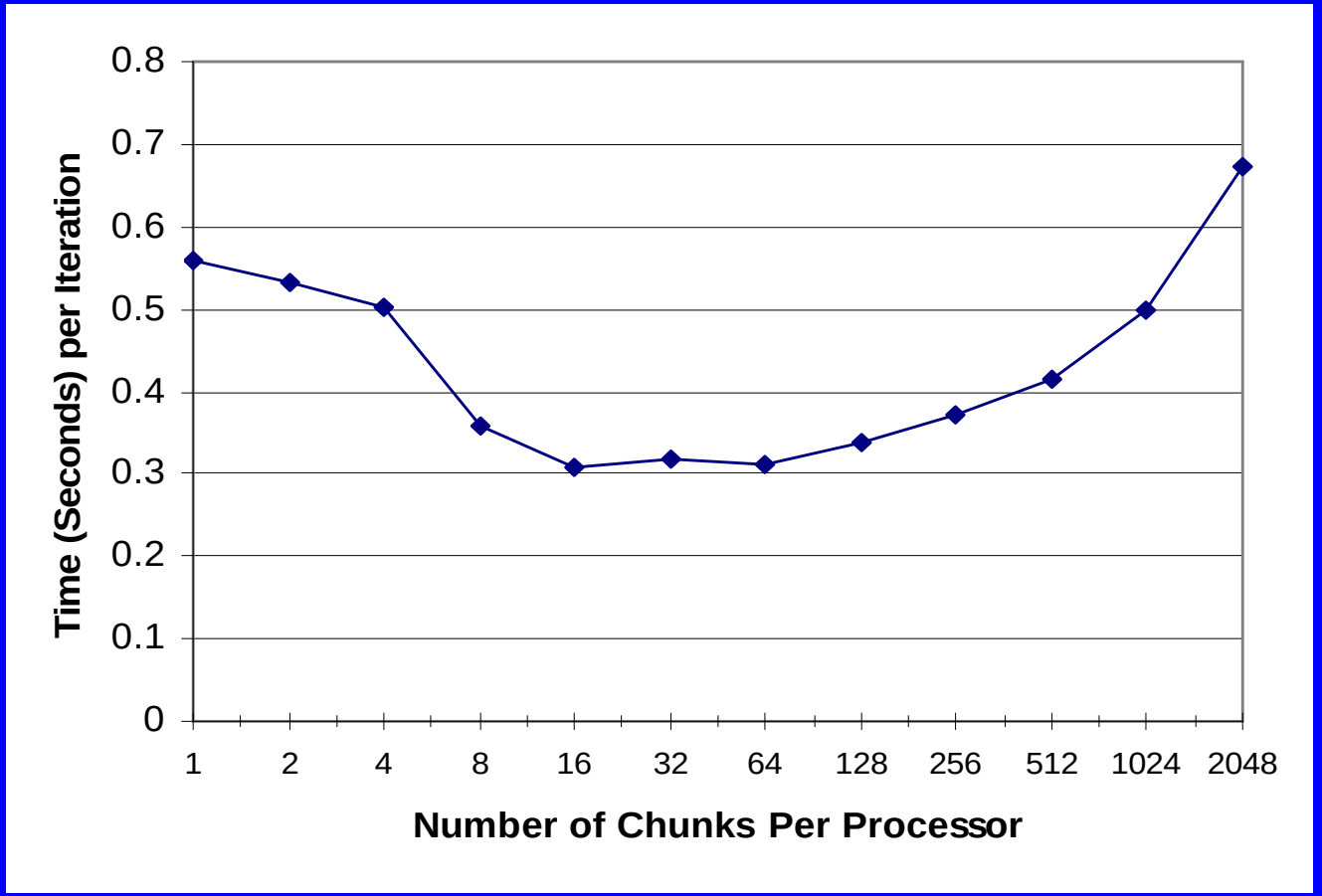
- The parallel-objects Runtime System can *observe, instrument, and measure* communication patterns
  - Communication is from/to objects, not processors
  - Load balancers use this to optimize object placement
  - Communication libraries can optimize
    - By substituting most suitable algorithm for each operation
    - Learning at runtime
  - E.g. Each to all individualized sends
    - Performance depends on many runtime characteristics
    - Library switches between different algorithms

V. Krishnan, MS Thesis, 1996

“Overhead” of Virtualization

Isn’t there significant overhead of virtualization?

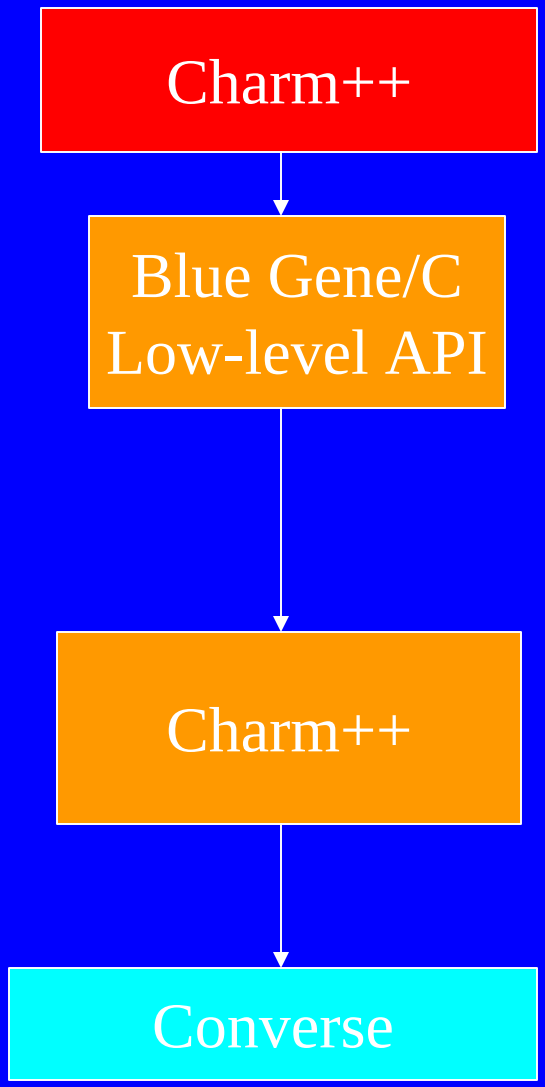
No! Not in most cases.

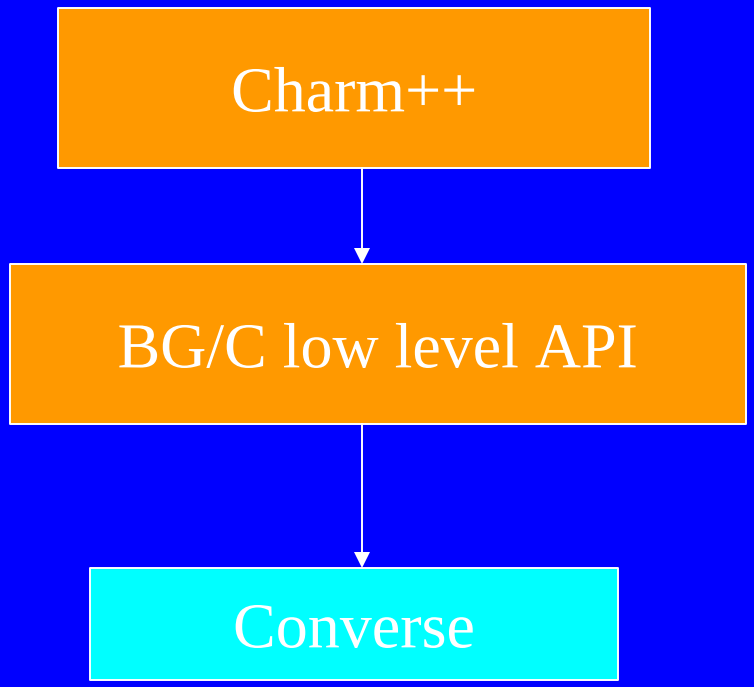


Using Charm++/AMPI for BG/L

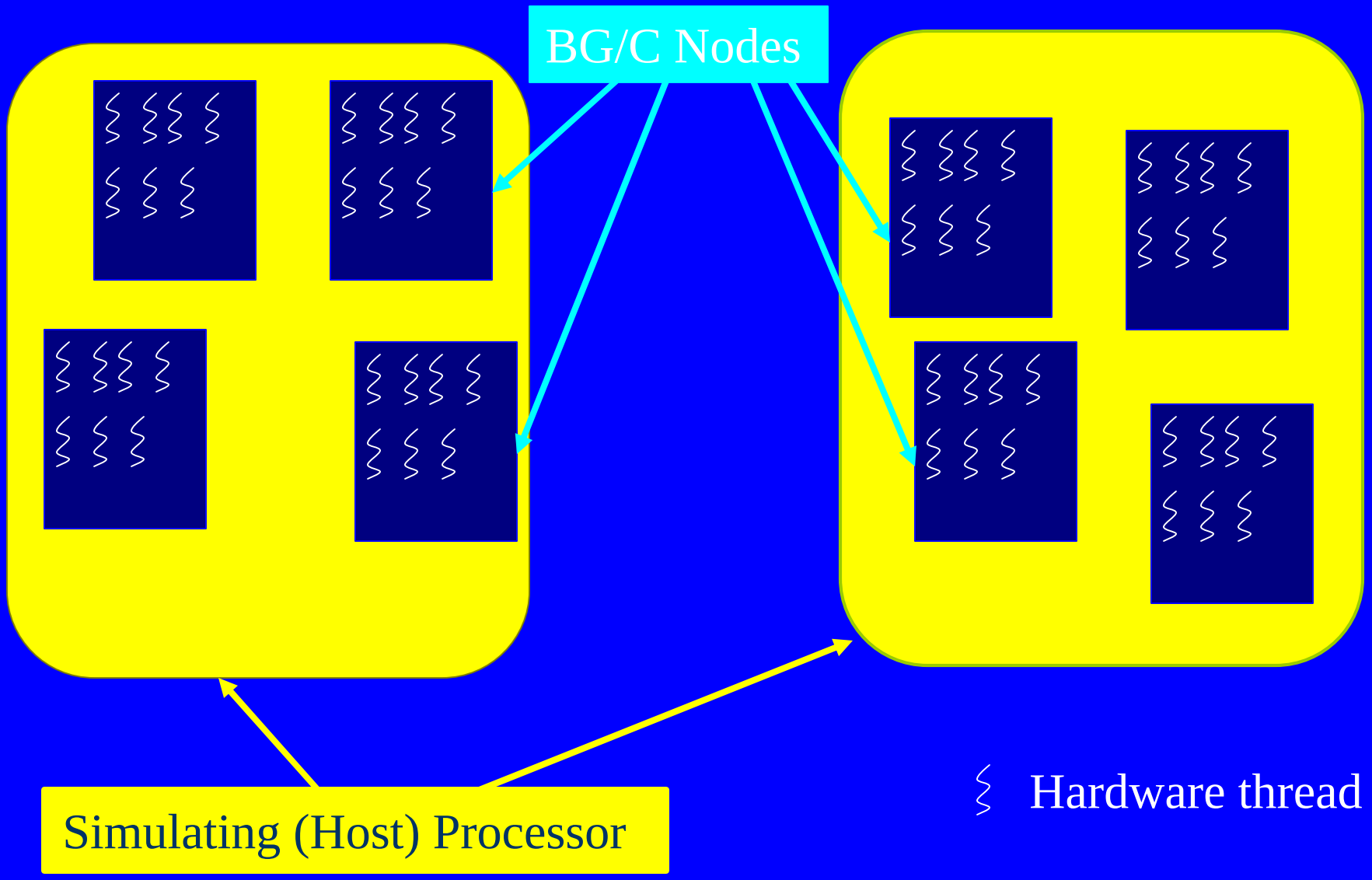
- How to develop any programming environment for a machine that isn't built yet
- Blue Gene/C emulator using charm++
  - Completed last year
  - Emulation runs on machines with hundreds of "normal" processors
- Recently retargeted and tested for BG/L
- Charm++ on Blue Gene Emulator

Structure of the Emulators





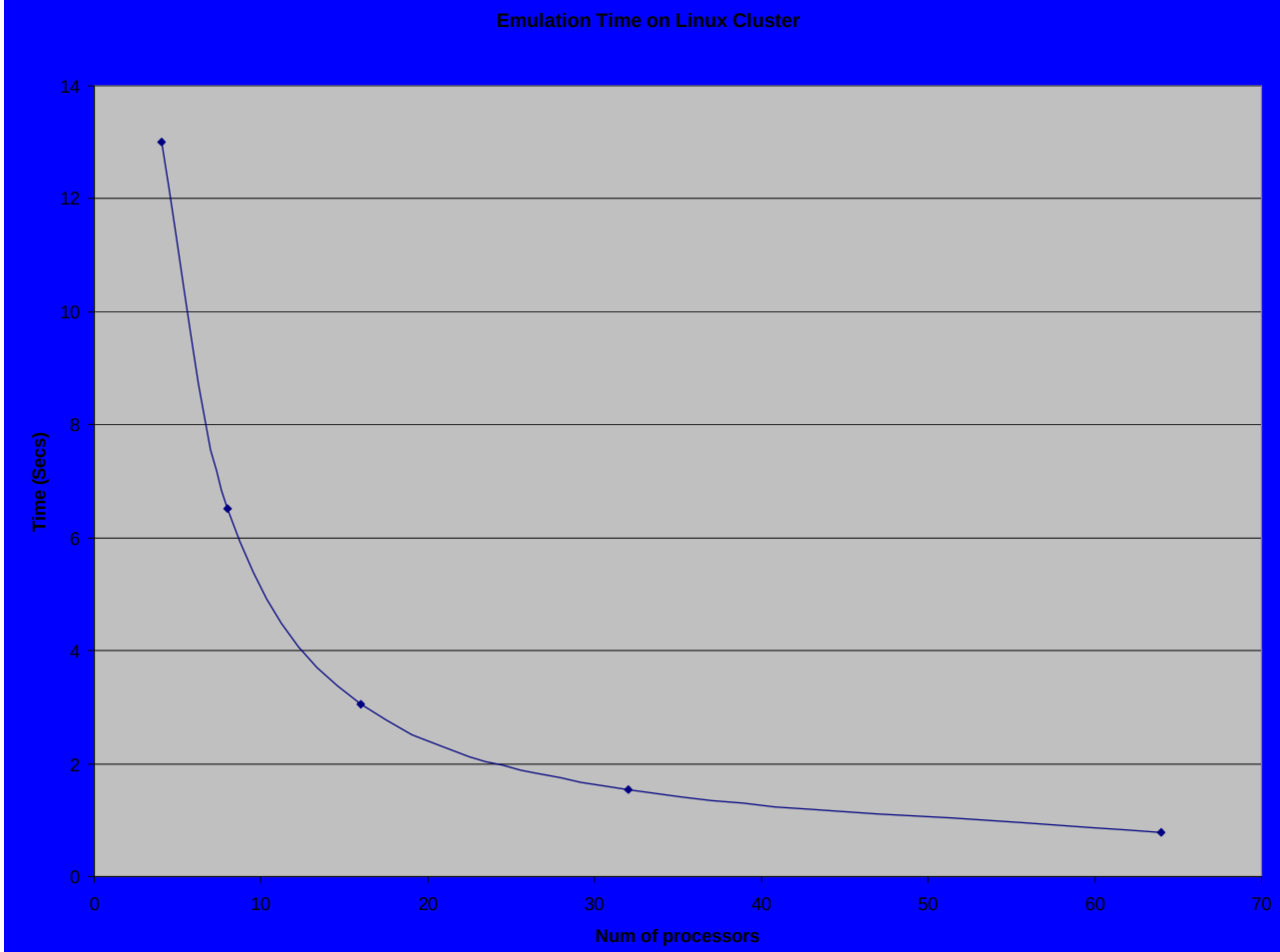
Emulation on a Parallel Machine



Emulation efficiency

- How much time does it take to run an emulation?
  - 8 Million processors being emulated on 100
  - In addition, lower cache performance
  - Lots of tiny messages
- On a Linux cluster, and Lemieux:
  - Emulation shows good speedup

Emulation efficiency



1000 BG/C nodes (10x10x10)

Each with 200 threads

(total of 200,000 user-level threads)

But Data is preliminary, based on one simulation

Emulator to Simulator

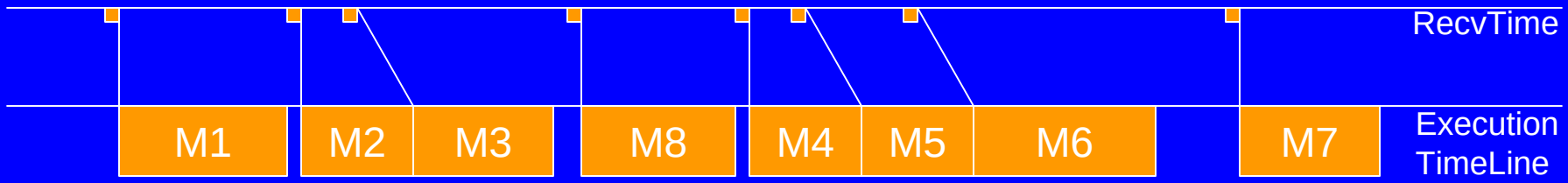
- Step 1: Coarse grained simulation
  - Simulation: performance prediction capability
  - Models contention for processor/thread
  - Also models communication delay based on distance
  - Doesn’t model memory access on chip, or network
  - How to do this in spite of out-of-order message delivery?
    - Rely on determinism of Charm++ programs
    - Time stamped messages and threads
    - Parallel time-stamp correction algorithm

Timestamp correction

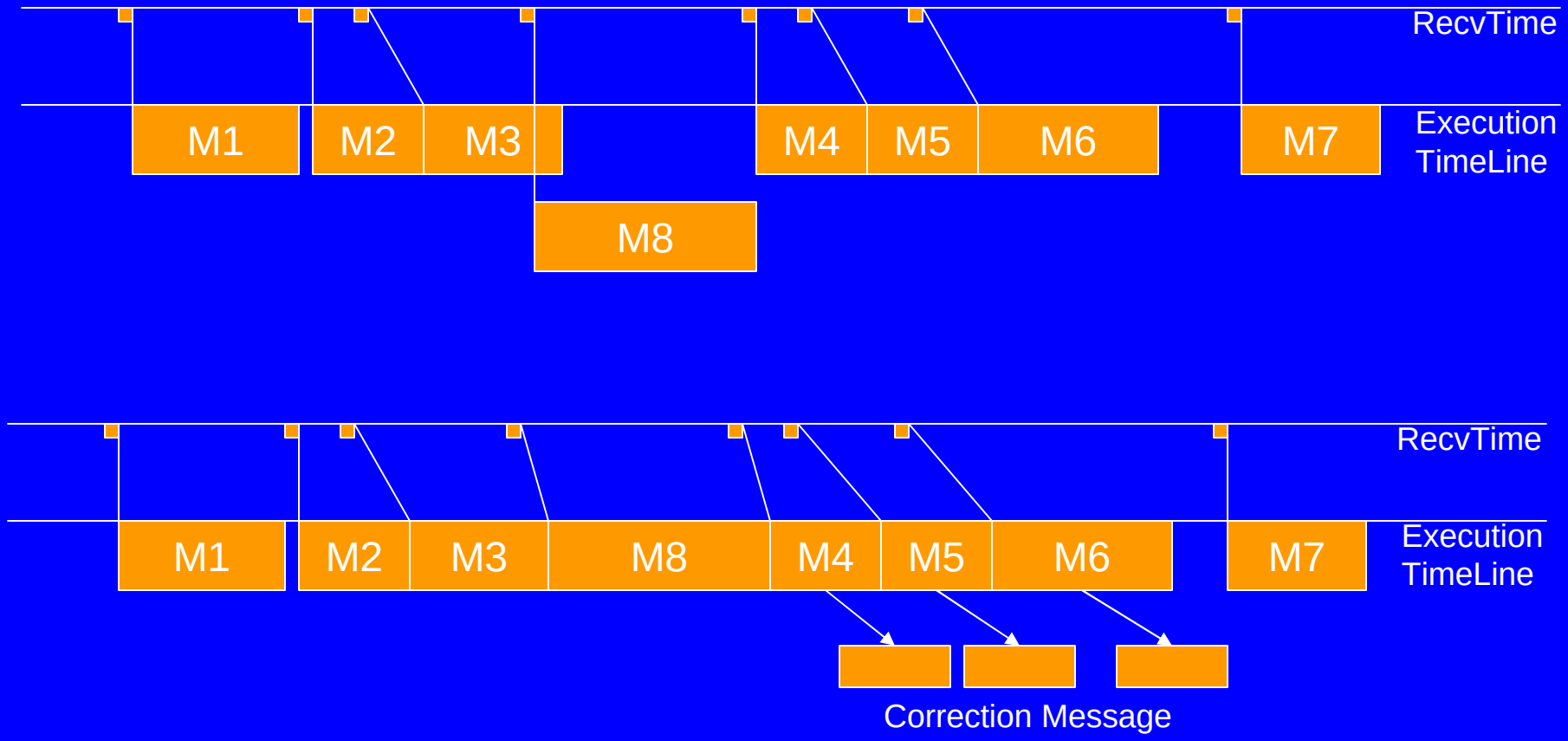
- Basic execution:
  - Timestamped messages
- Correction needed when:
  - A message arrives with an earlier timestamp than other messages “processed” already
- Determinism:
  - Messages to Handlers or simple objects
  - MPI style threads, without wildcard or irecvs
  - Charm++ with dependence expressed via structured dagger

Timestamps Correction

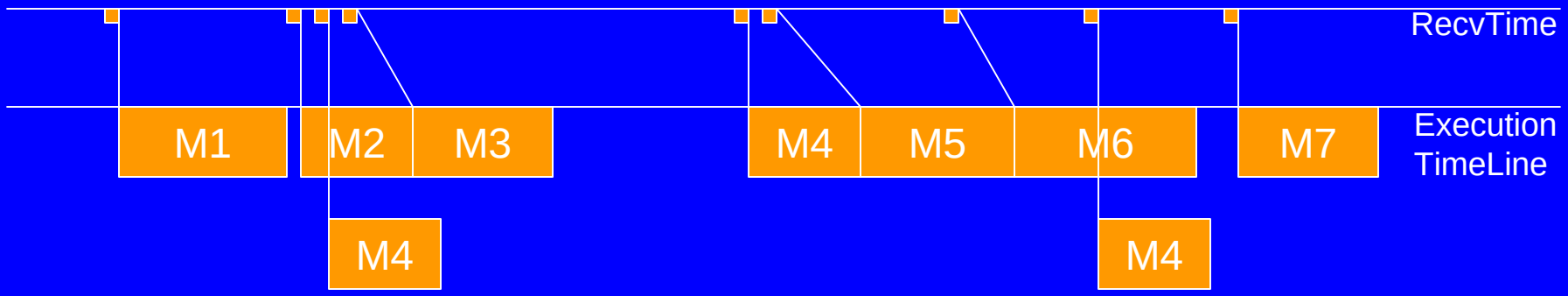
Timestamps Correction



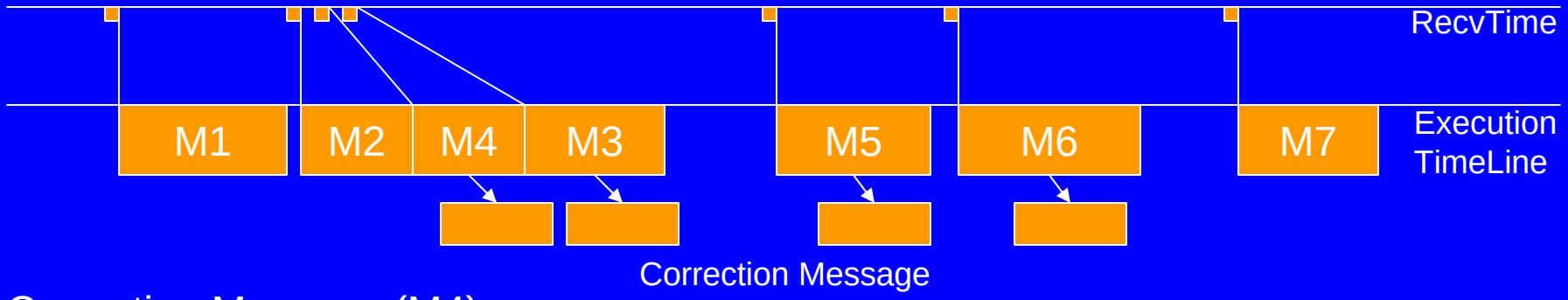
Timestamps Correction



Timestamps Correction



Correction Message (M4)



Correction Message (M4)

Performance of correction Algorithm

- Without correction
  - 15 seconds to emulate a 18msec timstep
  - 10x10x10 nodes with k threads each (200?)
- With correction
  - Version 1: 42 minutes per step!
  - Version 2:
    - “Chase” and correct messages still in queues
    - Optimize search for messages in the log data
    - Currently at 30 secs per step
- Alternative algorithm:
  - Trace-driven simulation

Emulator to Simulator

- Step 2: Add fine grained procesor simulation
  - Sarita Adve: RSIM based simulation of a node
    - SMP node simulation: completed
  - Also: simulation of interconnection network
  - Millions of thread units/caches to simulate in detail?
- Step 3: Hybrid simulation
  - Instead: use detailed simulation to build model
  - Drive coarse simulation using model behavior
  - Further help from compiler and RTS

Modeling layers

Applications

Libraries/RTS

Proc. Architecture

Network model

For each: need a detailed simulation and a simpler (e.g. table-driven) "model"

And methods for combining them

Applications on the current system

- Using BG Charm++
- LeanMD:
  - Research quality Molecular Dyanmics
  - Version 0: only electrostatics + van der Vaal
- Simple AMR kernel
  - Adaptive tree to generate millions of objects
    - Each holding a 3D array
  - Communication with “neighbors”
    - Tree makes it harder to find nbrs, but Charm makes it easy

Performance Issues and Techniques

- Scaling to 64K/128K processors
  - Communication
    - Bandwidth use more important than processor overhead
    - Locality:
  - Global Synchronizations
    - Costly, but not because it takes longer
    - Rather, small “jitters” have a large impact
    - Sum of Max vs Max of Sum
  - Load imbalance important, but so is grainsize
  - Critical paths

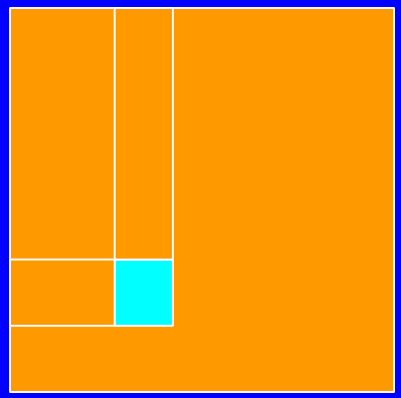
Parallelization Example: Molecular Dynamics in NAMD

- Collection of [charged] atoms, with bonds
  - Newtonian mechanics
  - Thousands of atoms (1,000 - 500,000)
  - 1 femtosecond time-step, millions needed!
- At each time-step
  - Calculate forces on each atom
    - Bonds:
    - Non-bonded: electrostatic and van der Waal's
  - Calculate velocities and advance positions
  - Multiple Time Stepping : PME (3D FFT) every 4 steps

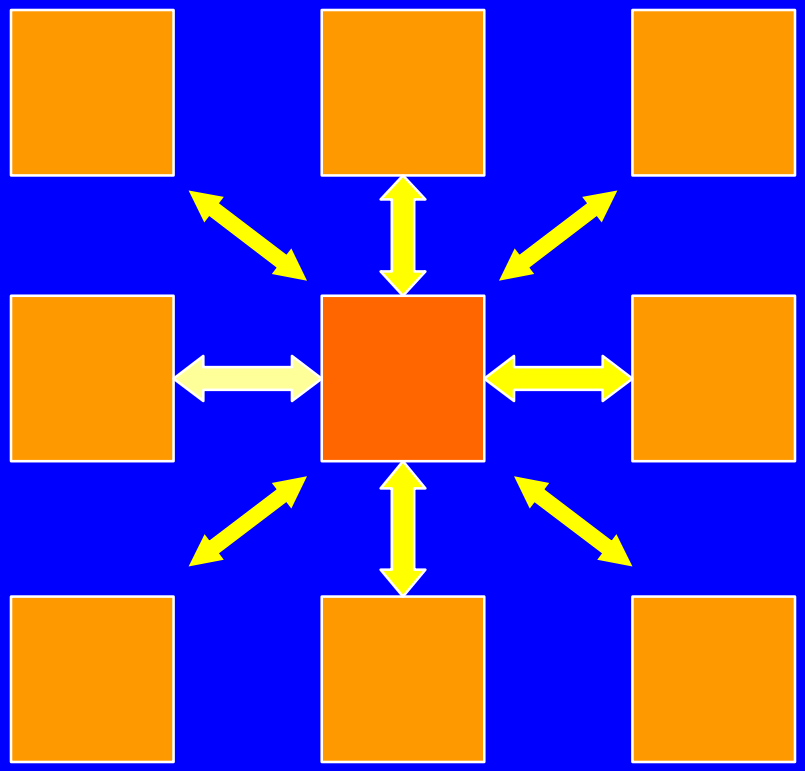
Collaboration with K. Schulten, R. Skeel, and coworkers

Traditional Approaches

- Replicated Data:
  - All atom coordinates stored on each processor
    - Communication/Computation ratio: P log P
- Partition the Atoms array across processors
  - Nearby atoms may not be on the same processor
  - C/C ratio: O(P)
- Distribute force matrix to processors
  - Matrix is sparse, non uniform,
  - C/C Ratio: sqrt(P)



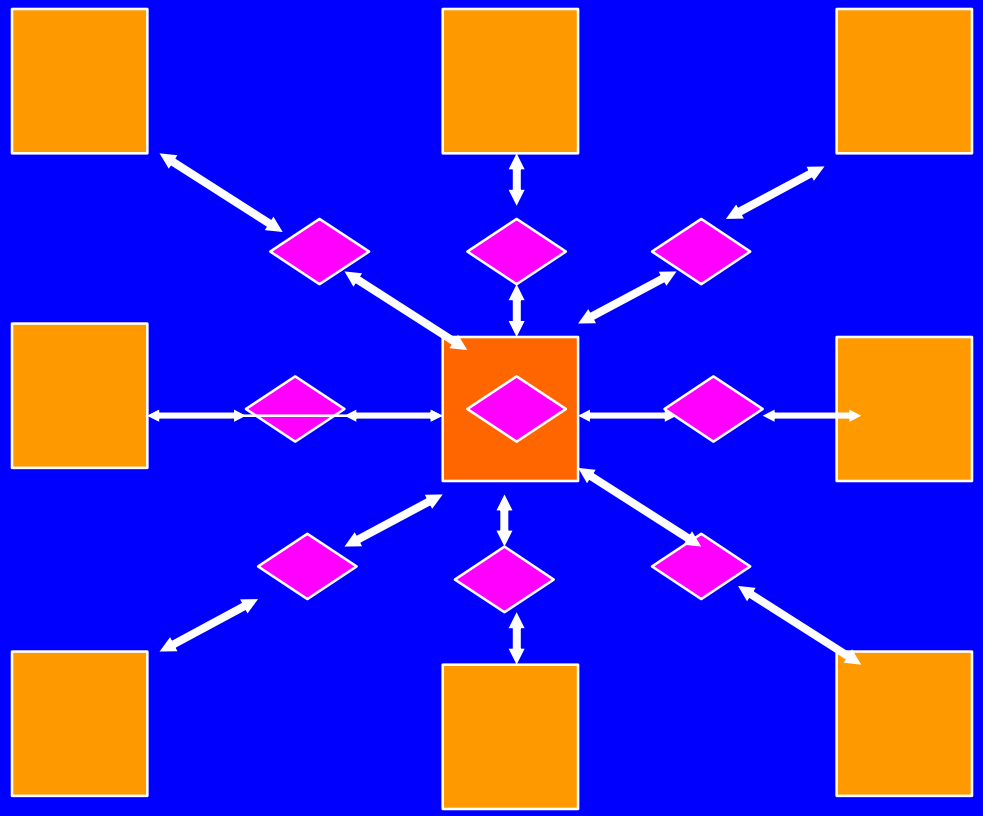
Spatial Decomposition



- C/C ratio: O(1)
- However:
  - Load Imbalance
  - Limited Parallelism

Object Based Parallelization for MD:

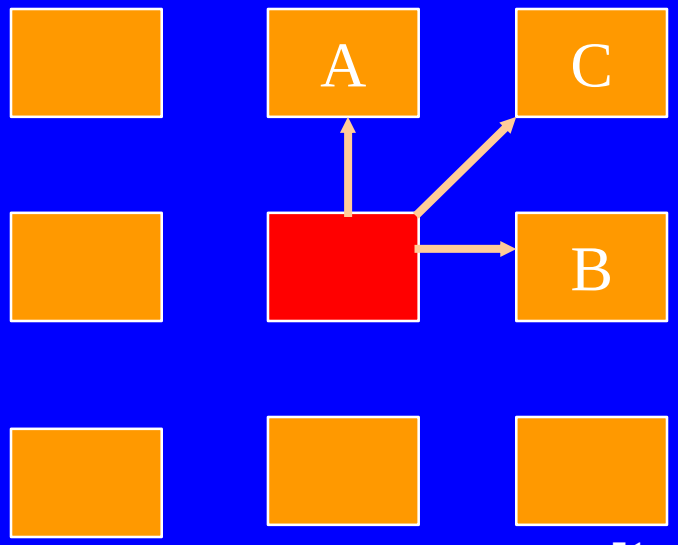
Force Decomposition + Spatial Deomp.

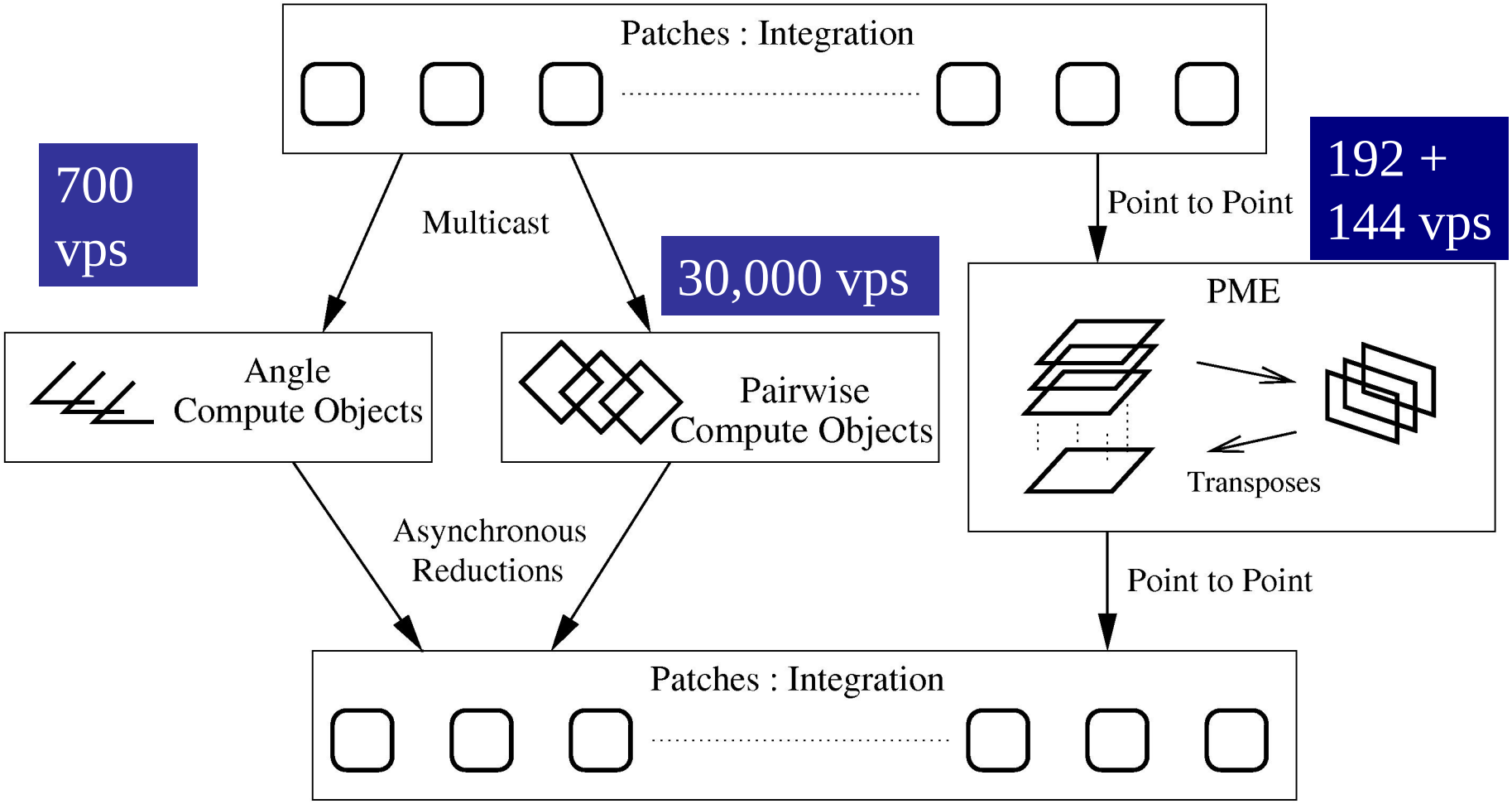


- Now, we have many objects to load balance:
  - –Each diamond can be assigned to any proc.
  - – Number of diamonds (3D):
  - –14·Number of Patches

Bond Forces

- Multiple types of forces:
  - Bonds(2), Angles(3), Dihedrals (4), ..
  - Luckily, each involves atoms in neighboring patches only
- Straightforward implementation:
  - Send message to all neighbors,
  - receive forces from them
  - 26*2 messages per patch!
- Instead, we do:
  - Send to (7) upstream nbrs
  - Each force calculated at one patch

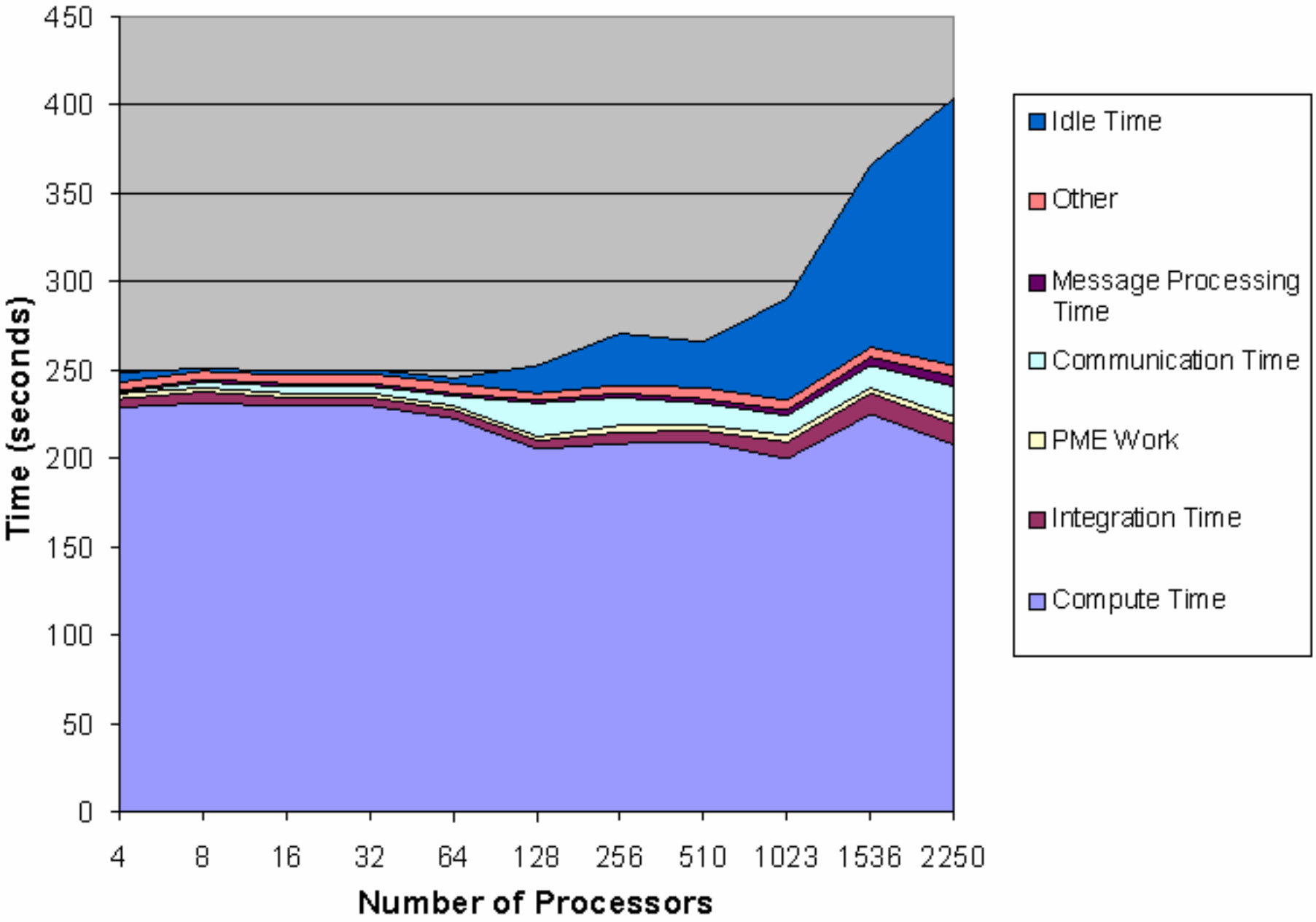



Patches : Integration
700 vps
Multicast
30,000 vps
Point to Point
192 + 144 vps
PME
Angle Compute Objects
Pairwise Compute Objects
Transposes
Asynchronous Reductions
Point to Point
Patches : Integration

NAMD performance using virtualization

- Written in Charm++
- Uses measurement based load balancing
- Object level performance feedback
  - using “projections” tool for Charm++
  - Identifies problems at source level easily
  - Almost suggests fixes
- Attained unprecedented performance

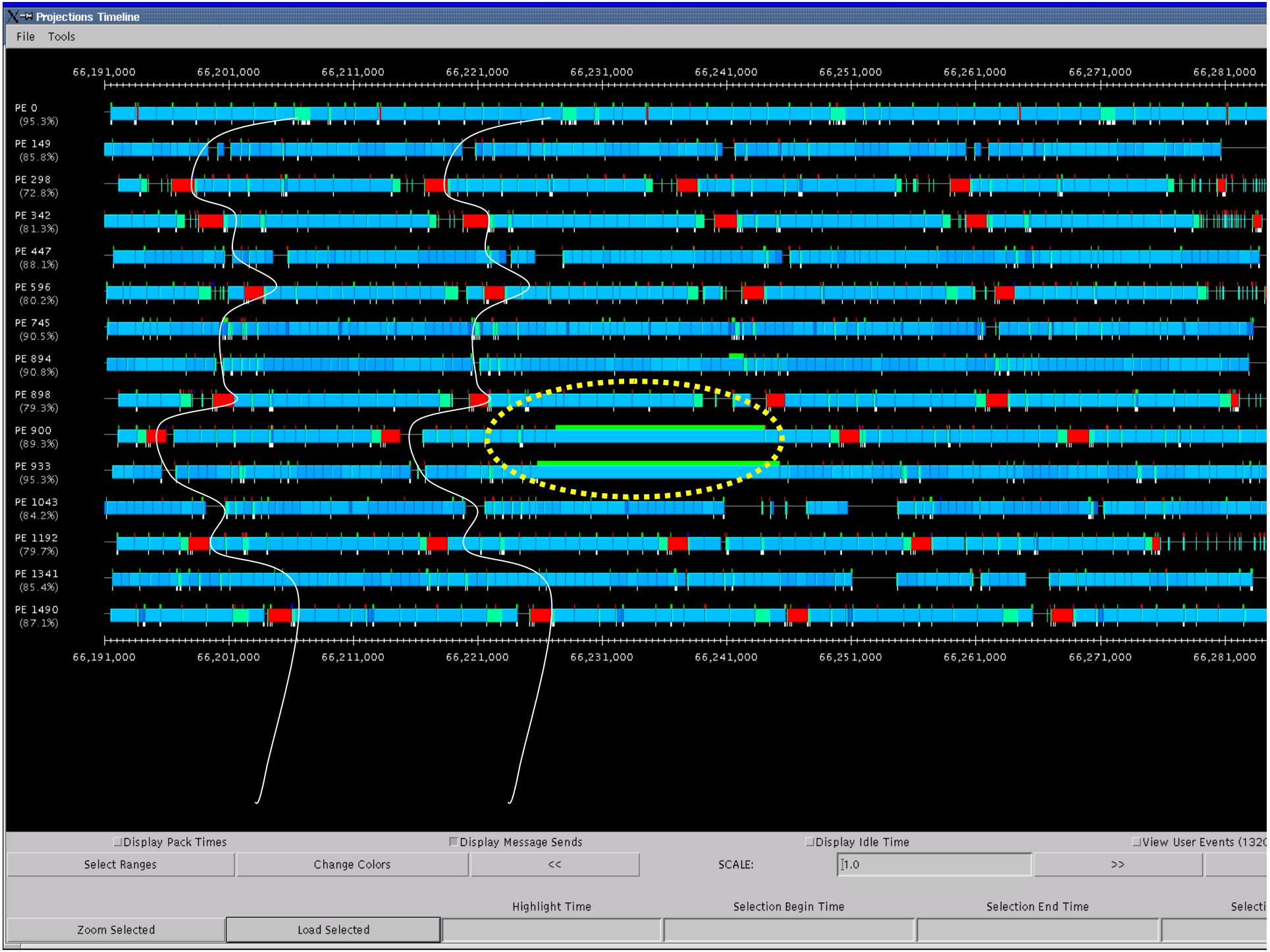
Relative Scaling

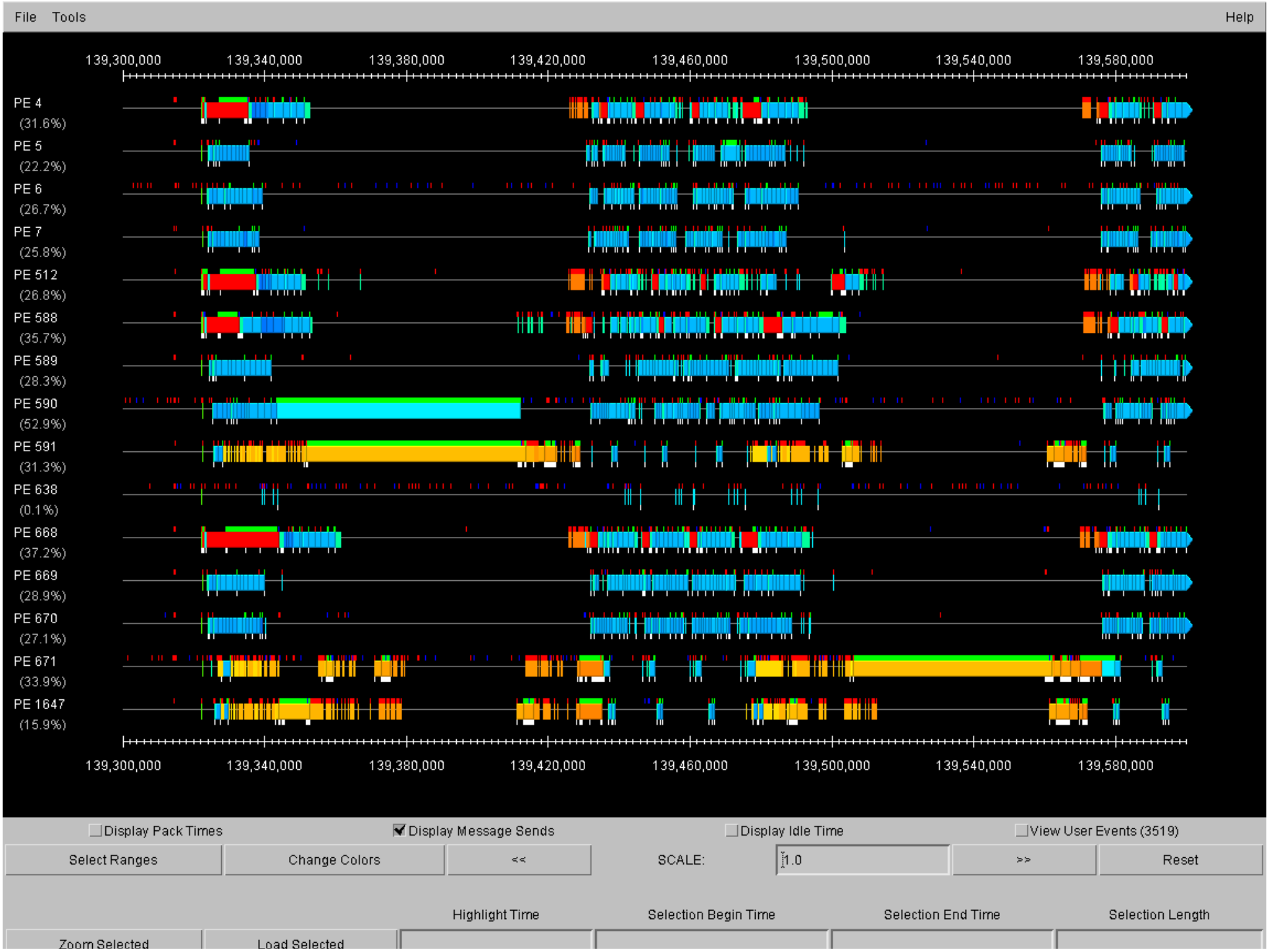


Performance: NAMD on Lemieux

		Time (ms)			Speedup			GFLOPS		
Procs	Per Node	Cut	PME	MTS	Cut	PME	MTS	Cut	PME	MTS
1	1	24890	29490	28080	1	1	1	0.494	0.434	0.48
128	4	207.4	249.3	234.6	119	118	119	59	51	57
256	4	105.5	135.5	121.9	236	217	230	116	94	110
512	4	55.4	72.9	63.8	448	404	440	221	175	211
510	3	54.8	69.5	63	454	424	445	224	184	213
1024	4	33.4	45.1	36.1	745	653	778	368	283	373
1023	3	29.8	38.7	33.9	835	762	829	412	331	397
1536	3	21.2	28.2	24.7	1175	1047	1137	580	454	545
1800	3	18.6	25.8	22.3	1340	1141	1261	661	495	605
2250	3	15.6	23.5	18.4	1599	1256	1527	789	545	733

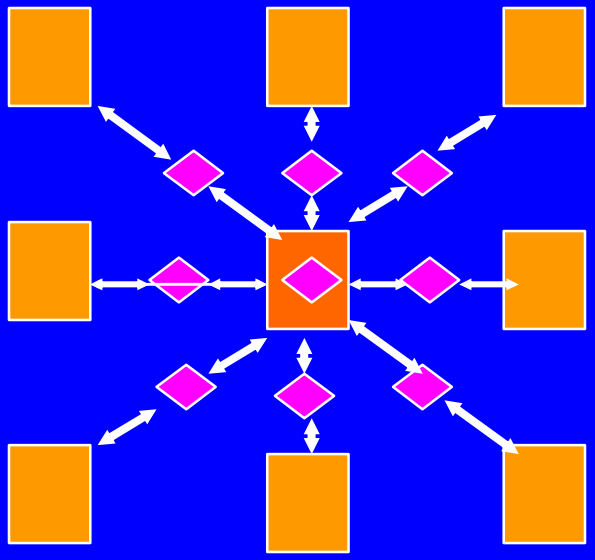
ATPase: 320,000+ atoms including water


Projections Timeline
File
Tools
66,191,000
66,201,000
66,211,000
66,221,000
66,231,000
66,241,000
66,251,000
66,261,000
66,271,000
66,281,000
PE 0 (95.3%)
PE 149 (85.8%)
PE 298 (72.8%)
PE 342 (81.3%)
PE 447 (88.1%)
PE 596 (80.2%)
PE 745 (90.5%)
PE 894 (90.8%)
PE 898 (79.3%)
PE 900 (89.3%)
PE 933 (95.3%)
PE 1043 (84.2%)
PE 1192 (79.7%)
PE 1341 (85.4%)
PE 1490 (87.1%)
Display Pack Times
Display Message Sends
Display Idle Time
View User Events (1320
Select Ranges
Change Colors
<<
SCALE:
1.0
>>
Highlight Time
Selection Begin Time
Selection End Time
Selecti
Zoom Selected
Load Selected

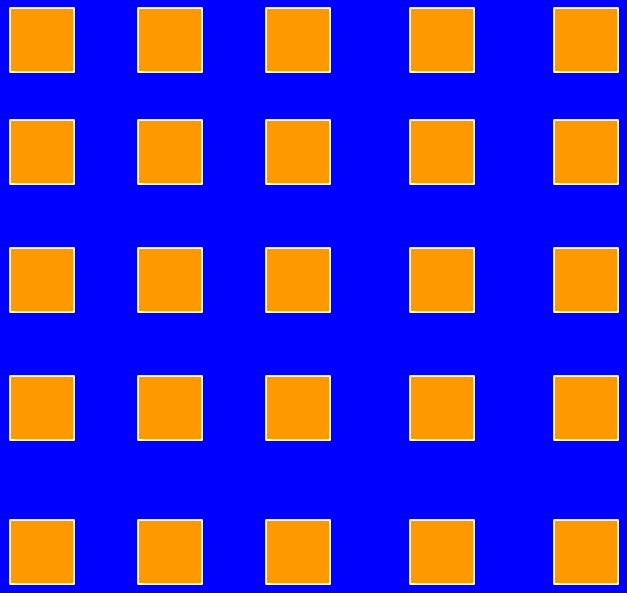

File
Tools
Help
139,300,000
139,340,000
139,380,000
139,420,000
139,460,000
139,500,000
139,540,000
139,580,000
PE 4
(31.6%)
PE 5
(22.2%)
PE 6
(26.7%)
PE 7
(25.8%)
PE 512
(26.8%)
PE 588
(35.7%)
PE 589
(28.3%)
PE 590
(52.9%)
PE 591
(31.3%)
PE 638
(0.1%)
PE 668
(37.2%)
PE 669
(28.9%)
PE 670
(27.1%)
PE 671
(33.9%)
PE 1647
(15.9%)
Display Pack Times
Display Message Sends
Display Idle Time
View User Events (3519)
Select Ranges
Change Colors
<<
SCALE:
1.0
>>
Reset
Highlight Time
Selection Begin Time
Selection End Time
Selection Length
Zoom Selected
Load Selected

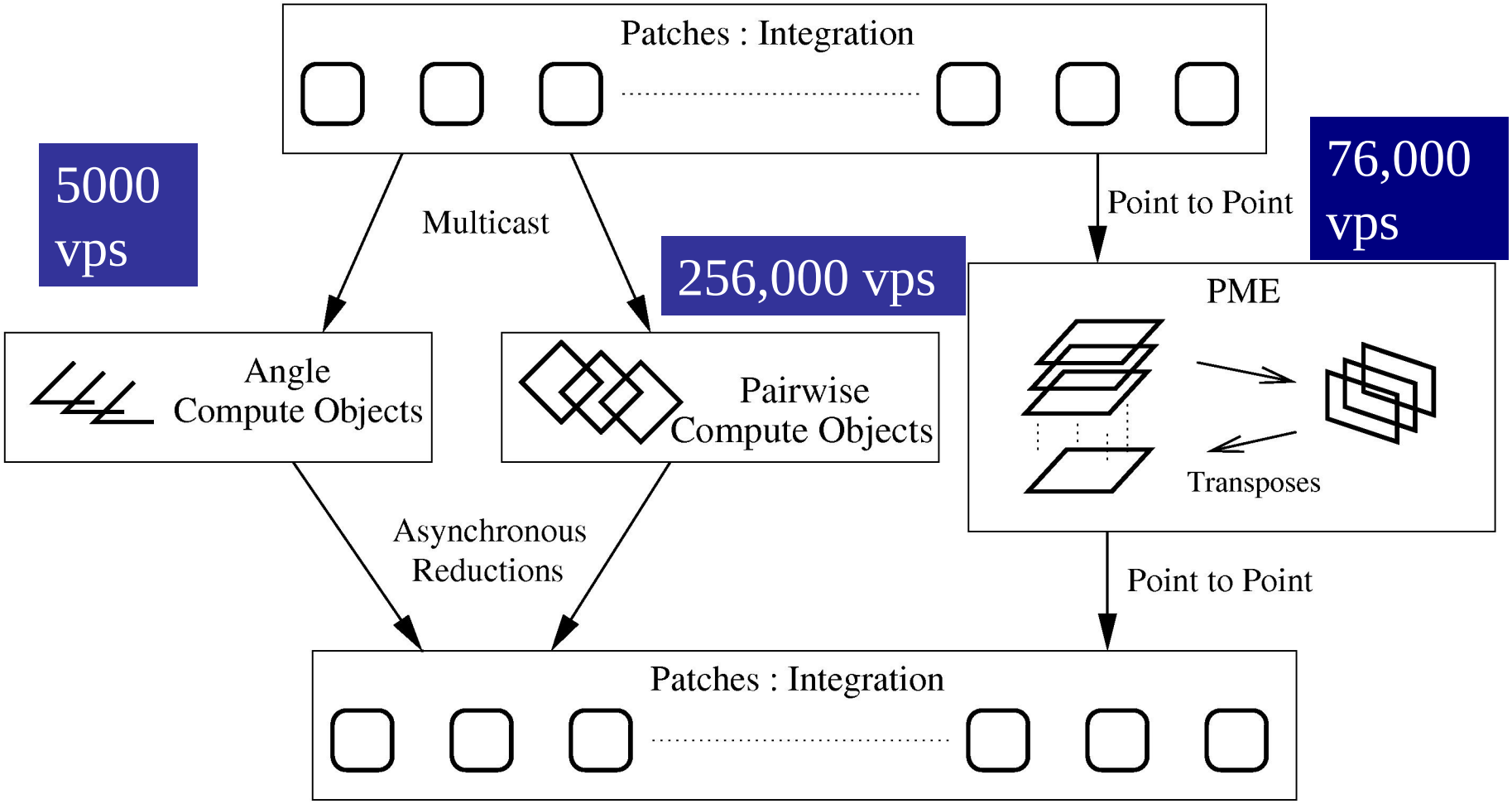
LeanMD for BG/L

- Need many more objects:
  - Generalize hybrid decomposition scheme
    - 1-away to k-away



2-away :
cubes are half the size.

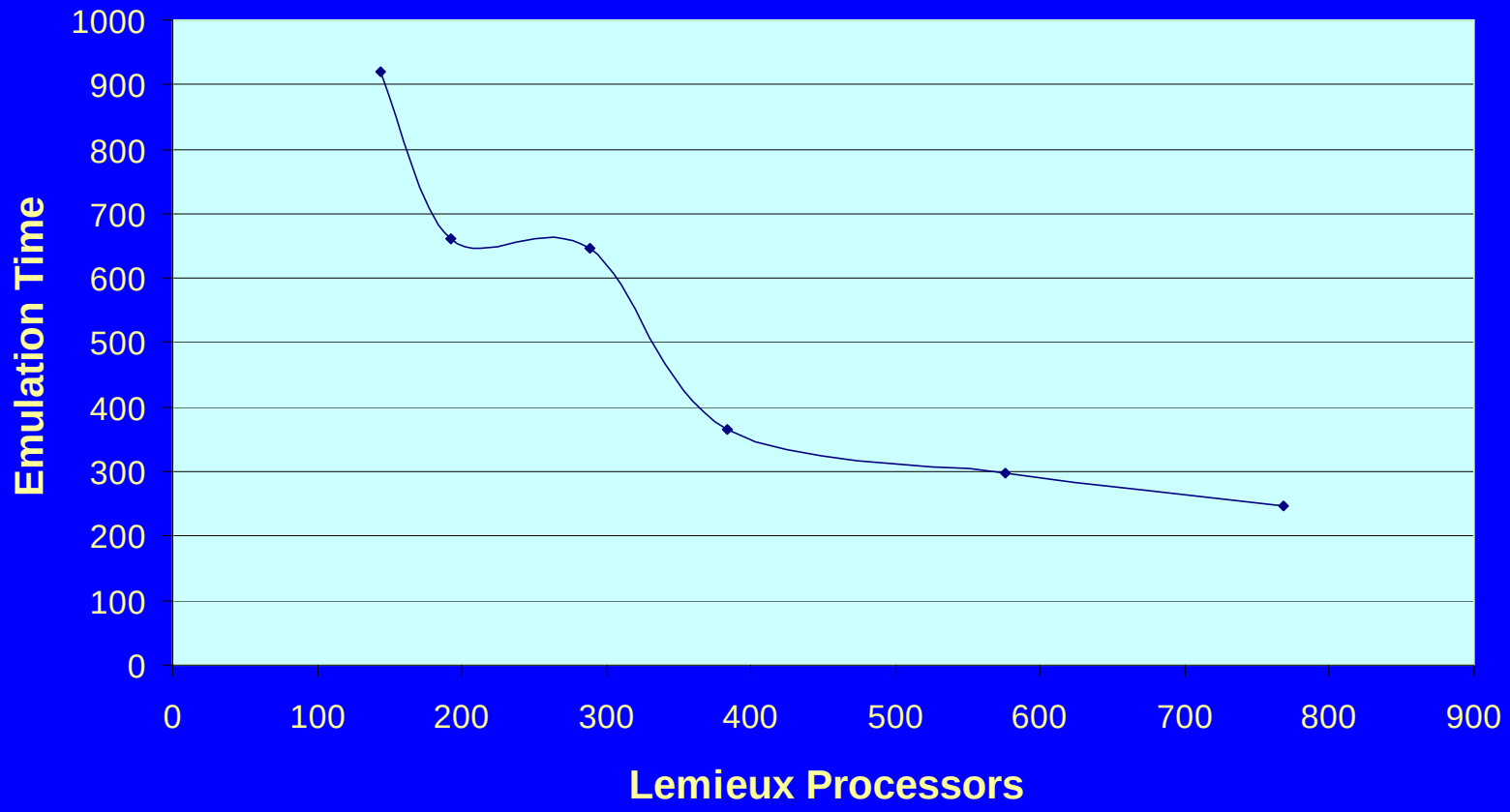



Patches : Integration
5000 vps
Multicast
256,000 vps
Point to Point
76,000 vps
PME
Angle Compute Objects
Pairwise Compute Objects
Transposes
Asynchronous Reductions
Point to Point
Patches : Integration

Emulation Speedup on Lemieux

32000 Node BG/L running LeanMD.

Emulation time per timestep



Ongoing Research

- Load balancing
  - Charm framework allows distributed and centralized
  - Recent years, we focused on centralized
    - Still ok for 3000 processors for NAMD
  - Reverting back to older work on distributed balancing
    - Need to handle locality of communication
      - Topology sensitive placement
    - Need to work with global information
      - Approx global info
      - Incomplete global info (only “neighborhood”)
    - Achieving global effects by local action…

Communication Optimizations

- Identify distinct communication patterns
  - Study different parallel algorithms for each
  - Conditions under which an algorithm is suitable
  - Incorporate algorithms and runtime monitoring into dynamic libraries
- Fault Tolerance
  - Much easier at object level: TMR, efficient variations
  - However, checkpointing used to be such an efficient alternative (low forward-path cost)
  - Resurrect past research

Multiparadigm programming

- Idea
- Converse
- Paradigms aimed at:
  - Charm++, MPI, AMPI, CRL,
  - Frameworks: FEM, AMR, Multiblock, particle
  - Other's paradigms:
    - HPF,
    - Virtualized versions of GA, UPC??,..

Summary

- Virtualization as a magic bullet: Charm/AMPI
  - Flexible and dynamic mapping to processors
  - Message driven execution:
    - Adaptive overlap, modularity, predictability
  - Principle of persistence
    - Measurement based load balancing,
    - Adaptive communication libraries
- BG/L Emulator and Simulator
  - Can run 128k proc. Program on current parallel m/cs
  - Charm++ and LeanMD ported to BG/L
  - Ongoing research in:
    - Scalable load balancing, communication optimizations,
    - Multiparadigm programming

More info:

http://charm.cs.uiuc.edu